Bolt Beranek and Newman Inc.

Report No. 4065

Research in Natural Language Understanding

Quarterly Progress Report No. 5, 1 September 1978 to 30 November 1978

Prepared for:
Advanced Research Projects Agency

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: BBN Report No. 4065
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): RESEARCH IN NATURAL LANGUAGE UNDERSTANDING, Quarterly Technical Progress Report No. 5, 1 Sep [illegible] — 30 Nov [illegible] 1978
5. TYPE OF REPORT & PERIOD COVERED: Quarterly Progress Report
6. PERFORMING ORG. REPORT NUMBER: BBN Report No. 4065

7. AUTHOR(s): Ronald J. Brachman
8. CONTRACT OR GRANT NUMBER(s): N00014-77-C-0378, ARPA Order-3414

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, MA 02138

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: (12) 34 p.

11. CONTROLLING OFFICE NAME AND ADDRESS:
Office of Naval Research
Dept. of the Navy
Arlington, VA 22217

12. REPORT DATE: 30 Nov [illegible] 1978
13. NUMBER OF PAGES: 23

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):

15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)
Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Semantic networks, knowledge representation, artificial intelligence, natural language conceptual structure, KLONE, memory models, property inheritance, structural description.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
One of the significant features of KLONE as a representation language is its facility for expressing structural interrelations among the conceptual subparts of an object, activity, or relationship. These interrelations are expressed as a set of "parameterized" relational and functional Concepts grouped together into structures called Structural Descriptions (SDs). Our abstract description of KLONE has included SDs from the beginning (see [Brachman, 1978] and [Woods and Brachman, 1978]). cont'd.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

060 100

20. Abstract (cont'd.)

in this quarter we have completed the design and coding of a set of INTERLISP functions to support their use in our language understanding system. This report presents the set of functions available to build, manipulate, query, and remove SDs. First we shall discuss SD structure and its relation to Concept structure, highlighting new features; then we shall present the complete set of functions with brief descriptions of their parameters, values, and operation.

RESEARCH IN NATURAL LANGUAGE UNDERSTANDING

Quarterly Technical Progress Report No. 5

1 September 1978 - 30 November 1978

ARPA Order No. 3414

Program Code No. 8D30

Name of Contractor:
Bolt Beranek and Newman Inc.

Effective Date of Contract:
1 September 1977

Amount of Contract:
$712,572

Contract No. N00014-77-C-0378

Contract Expiration Date:
31 August 1979

Short Title of Work:
Natural Language Understanding

Principal Investigator:
Dr. William A. Woods
(617) 491-1850, x4361

Scientific Officer:
Gordon D. Goldstein

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 3414

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-77-C-0378.

TABLE OF CONTENTS

page

Design and Implementation 1

I. SD Structure 2

II. Function Descriptions. 12

Design and Implementation of Structural Descriptions

One of the significant features of KLONE as a representation language is its facility for expressing structural interrelations among the conceptual subparts of an object, activity, or relationship. These interrelations are expressed as a set of "parameterized" relational and functional Concepts grouped together into structures called **Structural Descriptions** (SDs). Our abstract description of KLONE has included SDs from the beginning (see [Brachman, 1978] and [Woods and Brachman, 1978]) - in this quarter we have completed the design and coding of a set of INTERLISP functions to support their use in our language understanding system. This report presents the set of functions available to build, manipulate, query, and remove SDs. First we shall discuss SD structure and its relation to Concept structure, highlighting new features; then we shall present the complete set of functions with brief descriptions of their parameters, values, and operation.

I. SD Structure

A KLONE Generic Concept may have any number of SDs. An SD is indicated graphically as a diamond, to distinguish it from a Role, the other salient Concept subpart type. At the moment, there is only one type of SD (generic), and therefore only one type of Concept-SD eplink (see [Woods and Brachman, 1978]). Further, only Generic Concepts have SDs, although this will probably change once we understand the significance of relations among Instance Roles in Individual Concepts.*

One of the new features of the SD package is the ability to name SDs. Each SD may have attached to it a single user-specified atom, which is not constrained to be unique to that SD. Since an SD is intended to group together a set of relationships that are to hold among the fillers of the Concept's Roles, an SD name can be used to suggest the intent of the particular set of relationships. While KLONE itself makes no use of these names whatsoever, a user's program may take advantage of the facility to activate (i.e., to check as a predicate or to derive a Role filler) SDs of similar purpose (i.e., with the same name). Just as Role names are

* We expect to add particularized SDs for Individual Concepts in the next version of the system.

supported by the system, but not used by it, SD names hold no epistemological significance, but may be used by higher-level functions to group SDs for recognition, inference, etc.

An SD of a Generic Concept schematically expresses how the role fillers in an individuator of that Concept go together by way of a **set of ParaIndividuals.** A ParaIndividual is a parameterized version of a Generic Concept, unique to the Concept in whose SD it appears. It does not generally specify particular fillers for the Roles of the Generic (i.e., Individual Concepts), but instead "coreferences" Roles of its parent with Roles of the Concept in which it is used.* For example, in Fig. 1, we have the Generic Concept ARCH with a single SD called REQUIREMENTS. This SD has two ParaIndividuals, both derived from the Generic Concept SUPPORT. S1 coreferences the **supporter** Role of SUPPORT with the **left-upright** Role of ARCH, through the use of a **Coref Role.**** This means that, for every individuator of ARCH, there must exist an individuator of

* This is similar to a function call embedded within a function definition in LISP: the embedded (parameterized) call may have its arguments depend on the arguments of the enclosing function. Those arguments are bound only when the enclosing function is called (i.e., individuated).

** The Coref Role has a CorefSatisfies link to its parent Role (in this case the **supporter** Role of SUPPORT), and a CorefVal link to the Role from which the value is to be derived (**left-upright** here). This pair of links parallels closely the Satisfies/Val pair for Instance Roles, and performs the equivalent "parameterized" function.

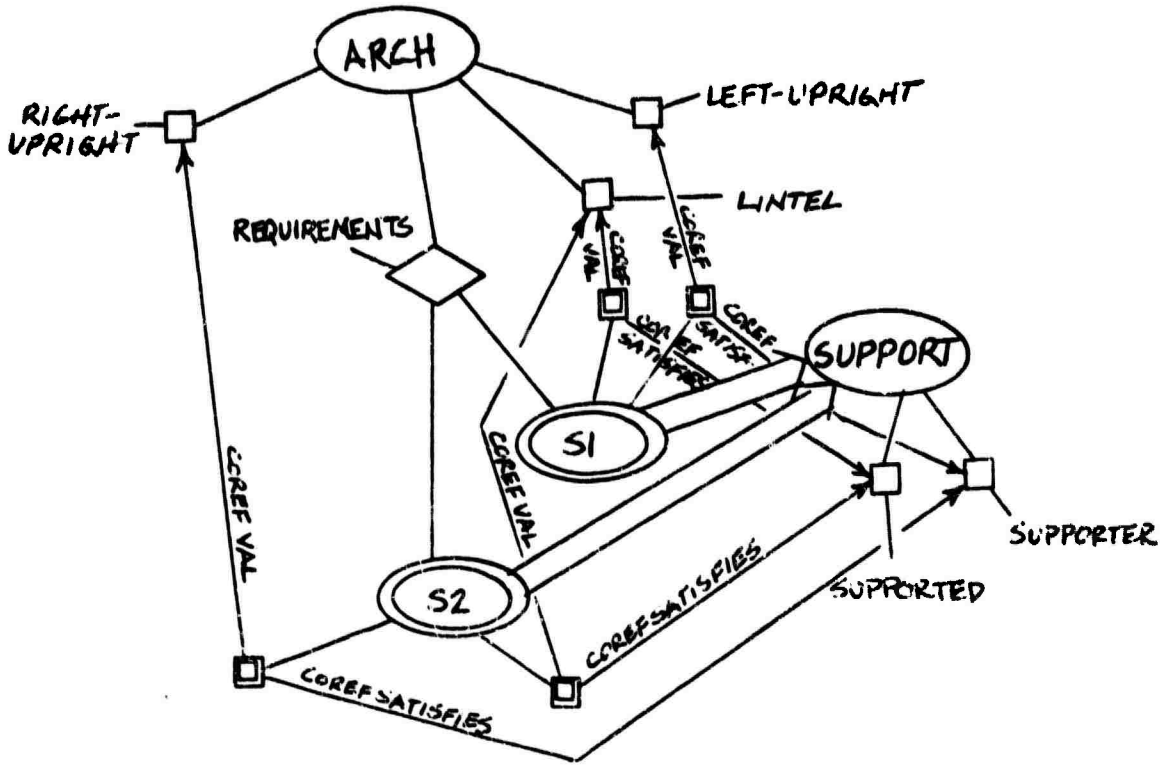

Fig. 1. ParaIndividuals.

SUPPORT whose **supporter** Role is to be filled by the filler of the **left-upright** Role of that ARCH. Since S1 also establishes a coreference between the **supported** Role of SUPPORT and the **lintel** Role of ARCH, S1 thus requires for any ARCH that its left upright supports its lintel. Similarly, S2 states that each ARCH must have its lintel supported by its right upright as well. In these cases, the Concept SUPPORT is not considered to be in the 'scope' of the SD - it has independent existence and can be referenced by other Concepts in the network. However, S1 and S2 are unique to ARCH,

and are not usable from outside, since their Roles depend on Roles of ARCH. In a sense, they are 'parameterized' by ARCH - for each individual ARCH, their values are uniquely determined (note the use of "its" above in describing S1 and S2).

In many cases, a ParaIndividual will have to access a subpart of an object. For example, say we wish to express that the name of an ARCH is the same as the last name of the PERSON that the ARCH is named after. In the configuration of Fig. 2, the obvious

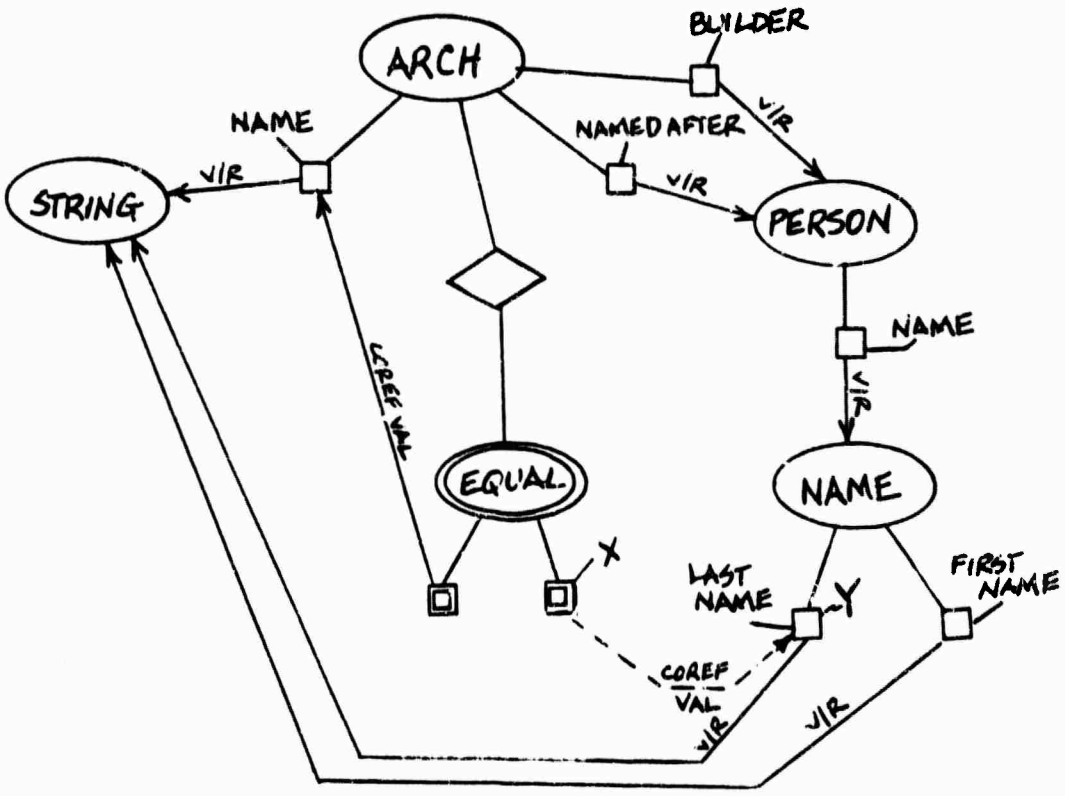

Fig. 2. Subpart access.

connection would be from the Coref Role, X, to the Role for the last name, Y. However, a single CorefVal pointer would not suffice: if any other PERSON or NAME were associated with ARCH, it would be impossible to tell which one was intended (for example, if ARCHes have **builders**, as in Fig. 2, a direct pointer to Y would not suffice to distinguish the builder's name from the named-After's name).

The problem here is that, while Roles are unique and distinguishable, their V/R's can be shared. Thus, a Role really has an implicit ParaIndividual (i.e., for the unique Role filler), and the V/R pointer to a Generic Concept is an abbreviation (to save the construction of a new Concept). There are two possible solutions to the ambiguous reference problem we might entertain: one is to provide an explicit ParaIndividual for each Generic Role, the other to allow a special kind of access link. In the current implementation, we have opted for the latter,* providing what we call a "Focus/SubFocus chain" or "Role Path". Fig. 3 illustrates such a link in the "namedAfter" case. In this case, the CorefVal pointer points to a structure which starts at a Role of the enclosing Concept (**namedAfter**), and walks down a chain of subpart Roles (**name**, **lastName**). This chain uniquely identifies the

* We are considering allowing the former in cases where it is needed. A decision on this will be reported in the next QPR.

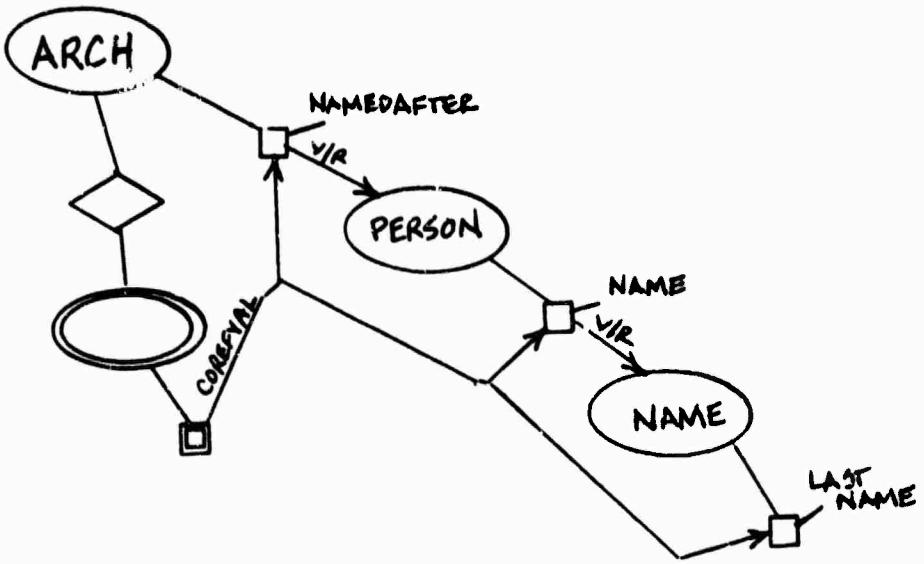

Fig. 3. A Role Path.

intended subpart, the **lastName** of the **name** of the **namedAfter** of the ARCH. These chains can be arbitrarily deep, as long as each Role pointed to is a Role of the V/R of the previous Role in the chain. In natural language, such a reference is expressed either as a string of "of" phrases, or as a string of possessives, e.g., the **ARCH's namedAfter's name's lastName.**

Since SDs are inherited through the Concept taxonomy, just as Roles are, we need to provide a "decentralized inheritance" mechanism (see [Brachman, 1977]) to allow some flexibility. At the moment, there is only a single way to relate an SD of a Concept to one (or more) of the SDs of its parent Concepts. This is the "Preempts" relationship, and is very simple - any SDs that would

normally be inherited but that instead are preempted by local SDs are not considered to be visible. This is similar to the Modifies relationship between Roles, although it is a bit more severe: nothing from the preempted SD is inherited. At the moment, we have no insights on further sophistication of inter-SD relationships because it is not clear how one might use some parts of an SD while overriding others. We must rely on the user to maintain a strict subcategorization hierarchy, since this cannot be guaranteed with such a simple overriding mechanism.

One further thing should be noted here about preempting SDs: the Preempts links form the inheritance backbone for SDNames. In this event, it is useful for an SD to be preempted multiply at the same Concept, so that the multiple preempting SDs can be considered "subSDs" of the one they preempt. An example will serve to illustrate this.

In Concept C3, a subConcept of both C1 and C2, we have three real SDs (S1, S2, S3), and one inherited one (S4). S1 and S2 are both subversions of the "INFER" SD of C1, presumably because there is some value in modularizing such a description. Meanwhile, S3 has two parent SDs, both of which are consequently not visible at C3. S3 would be considered both a "RECOG" and a "TEST" SD.

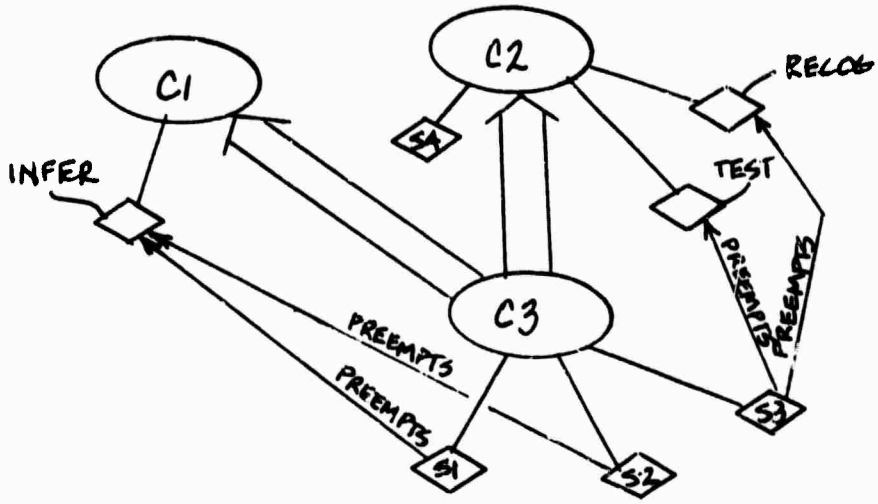

Fig. 4. SD Preempting.

Finally, we have provided a simple mechanism for specifying when SDs are to be "evaluated". Our intent with the SD facility is to provide a way of declaratively expressing constraints among the fillers of a Concept's Roles. An intelligent interpreter would presumably know when and how to take advantage of such neutral descriptions of relationships. At the moment, however, the KLONE system is not smart enough to successfully determine which SDs are relevant to checking structural validity, and which might be useful in deriving Role fillers that are dependent on others. We have for now provided a means of telling the interpreter directly which SDs (actually, which **parts** of SDs) are to be used as predicates to check the satisfaction of a generic description by an Individual Concept and which are to be used as functions in deriving values that are determined by Role fillers previously specified.

The mechanism is a simple one: one can specify for any ParaIndividual that it is to be included in the **Check**, **Derive**, or **NonActive** "facet" of its SD. This will support predicative and functional uses of SDs in this manner: all ParaIndividuals considered to be "Checks" are evaluated at individuation time. Individuators of those Concepts, with the appropriate Role bindings, are searched for, and if found, the Check is considered to succeed. "Deriving" ParaIndividuals will then be used to determine fillers for Roles of the Individual Concept that can be derived from the fillers of other roles. Individuators corresponding to the "Derive" ParaIndividuals are searched for (or "calculated", in a case like SUM or DISTANCE), where matches are attempted on all Roles **except for** those pointed to by CorefVal links that are considered **Derivable**. Those Roles are filled in by the corresponding values of the Individuator found in the Search.

All ParaIndividuals not explicitly marked for checking or deriving are considered "NonActive", and are only used either subordinately in checking or deriving (on "demand" by other ParaIndividuals, e.g., a ParaIndividual for DISTANCE would be NonActive if "used" by another ParaIndividual for SQUAREROOT, if what was being derived was the squareroot of the distance), or for inference and searching purposes.

References

Brachman, R.J. 1978. "A Structural Paradigm for Representing Knowledge," Ph.D. dissertation, Division of Engineering and Applied Physics, Harvard University. Also, BBN Report No. 3605, Bolt Beranek and Newman Inc., Cambridge, MA, May.

Woods, W.A. and Brachman, R.J. 1978. Research in Natural Language Understanding, Quarterly Technical Progress Report No. 1, 1 September 1977 to 30 November 1977, BBN Report No. 3742, Bolt Beranek and Newman Inc., Cambridge, MA, January. (Now available from NTIS as AD No. A053958).

II. Function Descriptions

In this section, we present a description of the set of functions now available in KLONE for manipulating SDs. Where relevant, we note changes or additions to the version of KLONE described in BBN Report No. 3848. The function descriptions start with the function name and an argument list, and then detail the types and meanings of the arguments. Finally, there appears a prose description of what the function does.

First, we present an indexed list of all of the function names in alphabetical order. The function descriptions follow, grouped roughly by type of action they perform.

SD FUNCTION DESCRIPTIONS

KLAddParaIndividual.....20
KLAddSD.....18
KLAddSDName.....19
KLChangeRoleCoref.....24
KLChangeSDName.....25
KLChangeSDOfParaIndividual.....26
KLCorefSatisfyRole.....21
KLDisEstablishAsPreemptor.....33
KLDisEstablishAsSDFacet.....29
KLEstablishAsPreemptor.....23
KLEstablishAsSDCheck.....27
KLEstablishAsSDDerive.....28
KLFindAllCorefRoles.....13
KLFindCorefSatisfiersOfRole.....14
KLFindNamedCorefRoles.....15
KLFindNamedSDs.....10
KLFindNamesOfSD.....12
KLFindOneNamedCorefRole.....16
KLFindParaIndividuators.....11
KLFindSDs.....9
KLFindValueForCorefInIndividuator.....17
KLGetConceptOfSD.....1
KLGetParaIndividualsInSD.....3
KLGetRoleCoref.....7
KLGetRoleCorefInverses.....8
KLGetSDChecks.....4
KLGetSDDerives.....5
KLGetSDOfParaIndividual.....6
KLGetSDs.....2
KLPreemptSD.....22
KLRemoveParaIndividual.....32
KLRemoveSD.....30
KLRemoveSDName.....31
KLSDP.....34

RETRIEVAL FUNCTIONS
========= =========

KLGetConceptOfSD [SD] (1)

description: Returns the Concept of which SD is an SD. An SD is an immediate part of only one Concept.

parameters: SD type: a single SD.

value: type: a single Concept.

KLGetSDs [Concept] (2)

description: Retrieves the set of SDs immediately encompassed by Concept. Does not do inheritance.

parameters: Concept type: a single Concept.

value: type: a set of SDs.

KLGetParaIndividualsInSD [SD] (3)
(** replaces KLGetParaIndividualInSDContext **)

description: Retrieves the set of ParaIndividuals occurring within a given SD.

parameters: SD type: a single SD.

value: type: a set of ParaIndividual Concepts.

KLGetSDChecks [SD] (4)

description: Returns the list of ParaIndividuals in the Check part of an SD.

parameters: SD type: a single SD.

value: type: either a set of ParaIndividual Concepts or NIL.

KLGetSDDerives [SD] (5)

description: Returns the list of ParaIndividuals in the Derive part of an SD.

parameters: SD type: a single SD.

value: type: either a set of ParaIndividual Concepts or NIL.

KLGetSDOfParaIndividual [ParaIndividual] (6)

description: Finds the SD in which a particular ParaIndividual Concept is situated. The SD is unique for a given ParaIndividual.

parameters: ParaIndividual type: a single ParaIndividual Concept.

value: type: a single SD.

KLGetRoleCoref [CorefRole] (7)

description: Given a Coref Role, returns its Coref Value -- a Role, Concept, or ParaIndividual.

parameters: CorefRole type: a single Coref Role.

value: type: a single CorefValue.

KLGetRoleCorefInverses [CorefValue] (8)

description: Retrieves all Coref Roles to which the value is bound as CorefValue.

parameters: CorefValue type: a single CorefValue.
meaning: the Concept or Role which is the CorefValue of the Roles to be found

value: type: either a set of Coref Roles or NIL.

KLFindSDs [Concept] (9)

description: Returns the complete set of SDs applicable at a Concept.

parameters: Concept type: a single Concept.

value: type: a set of SDs.

KLFindNamedSDs [Concept;SDName] (10)
(## new function ##)

description: Finds all inherited SDs at Concept that are named by SDName.

parameters: Concept type: a single Concept.
SDName type: a single SDName.

value: type: a set of SDs.

KLFindParaIndividuators [GenericConcept] (11)

description: Returns a list of all of the ParaIndividuators of a Concept. These include any ParaIndividuators of SubConcepts of the Concept, etc.

parameters: GenericConcept type: a single Generic Concept.

value: type: a set of ParaIndividual Concepts.

KLFindNamesOfSD [SD] (12)
(** new function **)

description: Finds the complete set of inherited names applying to SD. All names of preempted SDs, as well as a locally attached one, are included.

parameters: SD type: a single SD.

value: type: a set of SDNames.

KLFindAllCorefRoles [ParaIndividual] (13)

description: Finds all Coref Roles inherited by ParaIndividual.

parameters: ParaIndividual type: a single ParaIndividual Concept.

value: type: either a set of Coref Roles or NIL.

KLFindCorefSatisfiersOfRole [Concept;GenericRole] (14)
(** replaces KLFindCorefSpecializersOfRole **)

description: Finds all of the Instance Roles inheritable by Concept considered to be CorefSatisfiers of the one specified as argument. Includes those down Role inheritance chains.

parameters: Concept type: a single Concept.
meaning: the context in which the Coref Roles are to be found
GenericRole type: a single Generic Role.

value: type: either a set of Coref Roles or NIL.

KLFindNamedCorefRoles [ParaIndividual;RoleName] (15)

description: Finds all Coref Roles of ParaIndividual that are considered to be named by RoleName.

parameters:			
	ParaIndividual	type:	a single ParaIndividual Concept.
	RoleName	type:	a single RoleName.
value:		type:	either a set of Coref Roles or NIL.

KLFindOneNamedCorefRole [ParaIndividual;RoleName] (15)

description: Finds the one Coref Role of ParaIndividual that is considered to be named by RoleName.

parameters:			
	ParaIndividual	type:	a single ParaIndividual Concept.
	RoleName	type:	a single RoleName.
value:		type:	either a single Coref Role or NIL.

KLFindValueForCorefInIndividuator [CorefRole;IndividualConcept] (17)
(** new function **)

description: Finds the value in a particular Individual Concept that is indicated by a Coref Role in one of its inherited SD's. That is, uses IndividualConcept as the parameter for a ParaIndividuator, and evaluates a coreference in that context.

parameters:			
	CorefRole	type:	a single Coref Role.
		meaning:	some Coref Role in an inherited SD of IndividualConcept
	IndividualConcept	type:	a single Individual Concept.
value:		type:	a single RoleValue.
		meaning:	the meaning of the coreference in the context of IndividualConcept

STRUCTURE BUILDING FUNCTIONS

KLAddSD [GenericConcept;SDName] (18)

description: Adds a blank SD to an existing Concept.

parameters:	GenericConcept	type:	a single Generic Concept.
	SDName{optional}	type:	a single SDName.
		meaning:	if specified, atatched to new SD
value:		type:	a single SD.
		meaning:	the new SD created as a part of GenericConcept

KLAddSDName [SD;SDName] (19)
(** new function **)

description: Adds a name to an SD - an SD can have only one name directly attached (although it can inherit others).

parameters:	SD	type:	a single SD.
	SDName	type:	a single SDName.

KLAddParaIndividual [SD;ConceptName;GenericConcept;WiringList] (20)

description: Creates a new ParaIndividual Concept, and places it within a Structural Description of an existing Concept. GenericConcept is the Concept that the new one ParaIndividuates, and the wiring list specifies how each of its Roles are to be matched up and filled. The ParaIndividual added to the SD is not considered initially to be in the Check or Derive parts.

parameters:	SD	type:	a single SD.
		meaning:	SD to which ParaIndividual is added
	ConceptName(optional)	type:	a single ConceptName.
		meaning:	name of new ParaIndividual
		restrictions:	(~(KLGetNamedConcept ConceptName))
	GenericConcept	type:	a single Generic Concept.
		meaning:	Concept being ParaIndividuated
	WiringList	type:	a set of triples of a Generic Role, one of 'CorefSatisfies or 'Satisfies, and a Value.
		meaning:	the set of Role-Role connections for constructing the ParaIndividual's parts
value:		type:	a single ParaIndividual Concept.
		meaning:	the newly created ParaIndividual

KLCorefSatisfyRole [ParaIndividual;CorefValue;SuperRole] (21)
(** replaces KLAddCorefRole **)

description: Adds a Coref Role to an existing Concept, expected to be a ParaIndividual. SuperRole is a Role of the Concept being ParaIndividuated, and CorefValue is the binding of that Role in the context of ParaIndividual. CorefValues are 1: Roles of the Concept in whose SD the ParaIndividual lies, 2: that Concept itself, which means me -- the particular Individual Concept that has this SD inherited and is invoking it, 3: a Coref Role of some other ParaIndividual in the same SD as the ParaIndividual, 4: some other ParaIndividual in the same SD, 5: a list of Roles such that the first is a Role of the enclosing Concept and each thereafter is a Role of the V/R of the preceding one -- this is a Focus/SubFocus chain.

parameters:	ParaIndividual	type:	a single ParaIndividual Concept.
		meaning:	Concept to which the new Role is to be added
	CorefValue	type:	a single CorefValue.
		meaning:	Source of the value of the Role when an individuator is finally constructed. If a Role, then must be of the Concept in whose SD ParaIndividual appears, or a Role of another ParaIndividual in the same SD; if a Concept, must be another ParaIndividual, in same SD as ParaIndividual, or the enclosing Concept itself; if a list, must be a list of Roles accessible by walking down Roles from the enclosing Concept -- i.e., Focus/SubFocus
	SuperRole	type:	a single Generic Role.
		meaning:	Role that is being CorefSatisfied
value:		type:	a single Coref Role.
		meaning:	the new Coref Role that is created as part of ParaIndividual

KLPreemptSD [GenericConcept;SD;SDName] (22)

description: Adds a new SD to GenericConcept that will override SD, whereas SD would normally be expected to be inherited directly by GenericConcept. SDName is an optional name to be attached to the new SD.

parameters:			
	GenericConcept	type:	a single Generic Concept.
		meaning:	a concept that would normally inherit SD, if it weren't preempted
	SD	type:	a single SD.
		meaning:	the SD being preempted
	SDName(optional)	type:	a single SDName.
value:		type:	a single SD.
		meaning:	the new SD added to GenericConcept

KLEstablishAsPreemptor [PreemptingSD;SuperSD] (23)

description: Establishes a relationship between two SD's wherein one (the PreemptingSD) overrides the other. SuperSD would normally be inherited intact by the Concept in which PreemptingSD appears, but ceases to have effect after the establishing.

parameters:			
	PreemptingSD	type:	a single SD.
		meaning:	the SD that will override an inherited one
	SuperSD	type:	a single SD.
		meaning:	the SD that will no longer be in effect

CHANGING FUNCTIONS
======== =========

KLChangeRoleCoref [CorefRole;NewCorefValue] (24)

description: Deletes the old CorefValue of a Coref Role, and replaces it with a new one. Does not activate Delete or Add IHooks.

parameters:			
	CorefRole	type:	a single Coref Role.
	NewCorefValue	type:	a single CorefValue.
value:		type:	a single CorefValue.
		meaning:	the CorefValue of Role after the change

KLChangeSDName [SD;NewSDName] (25)
(** new function **)

description: Removes old name attached to SD and replaces it with NewSDName.

parameters:			
	SD	type:	a single SD.
	NewSDName	type:	a single SDName.

KLChangeSDOfParaIndividual [ParaIndividual;NewSD] (26)
(** new function **)

description: This function allows a ParaIndividual to be transferred from one SD to another. The SDs must be of the same Concept, and no CorefRole of the ParaIndividual can point to anything within any SD except the new one.

parameters: ParaIndividual type: a single ParaIndividual Concept.
NewSD type: a single SD.

SD CHECK/DERIVE FUNCTIONS

KLEstablishAsSDCheck [ParaIndividual] (27)

description: Takes a ParaIndividual that is part of an SD and puts it in the Check part of that SD, so that it will be used as a predicate when the enclosing Concept is individuated.

parameters: ParaIndividual type: a single ParaIndividual Concept.

KLEstablishAsSDDerive [ParaIndividual] (28)

description: Takes a ParaIndividual that is part of an SD and puts it in the Derive part of that SD, so that it will be used as a function to derive new Role fillers when the enclosing Concept is individuated.

parameters: ParaIndividual type: a single ParaIndividual Concept.

KLDisEstablishAsSDFacet [ParaIndividual] (29)
(** replaces KLDisEstablishAsSDCheck and KLDisEstablishAsSDDerive **)

description: Removes a ParaIndividual from either the Check or Derive part of an SD. Leaves the ParaIndividual as part of the SD.

parameters: ParaIndividual type: a single ParaIndividual Concept.

REMOVING/DELETING FUNCTIONS
======== ======== =========

KLRemoveSD [SD] (30)

description: Removes an SD from a Concept, throwing away any ParaIndividuals that it comprises.

parameters: SD type: a single SD.

KLRemoveSDName [SD] (31)
(** new function **)

description: Removes any name attached to an SD - does not affect inherited names.

parameters: SD type: a single SD.

KLRemoveParaIndividual [ParaIndividual] (32)

description: Removes a ParaIndividual from an SD, and throws away all of its connecctions to parts of the Concept it was in.

parameters: ParaIndividual type: a single ParaIndividual Concept.

KLDisEstablishAsPreemptor [SD] (33)

description: Removes the preemptive relationship between SD and an SD of a SuperConcept, wherein SD had previously overridden the one in the SuperConcept. The SD that was previously preempted will now be inherited intact by the Concept in which SD appears.

parameters: SD type: a single SD.
meaning: an SD that will no longer preempt one inherited from a SuperConcept of its enclosing Concept.

OTHER FUNCTIONS
===== =========

KLSDP [Anything] (34)

description: Predicate to test whether an item is an SD. If so, the item is returned; if not, NIL is returned.

parameters: Anything type: anything.

value: type: either NIL or a SD.
meaning: if non-NIL, returns the SD passed in as argument

Official Distribution List
Contract N00014-77-C-0378

	Copies
Defense Documentation Center Cameron Station Alexandria, VA 22314	12
Office of Naval Research Information Systems Program Code 437 Arlington, VA 22217	2
Office of Naval Research Code 200 Arlington, VA 22217	1
Office of Naval Research Code 455 Arlington, VA 22217	1
Office of Naval Research Code 458 Arlington, VA 22217	1
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, MA 02210	1
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, IL 60605	1
Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, CA 91106	1
Office of Naval Research New York Area Office 715 Broadway - 5th Floor New York, NY 10003	1

Naval Research Laboratory 6
Technical Information Division
Code 2627
Washington, D.C. 20380

Naval Ocean Systems Center 1
Advanced Software Technology Division
Code 5200
San Diego, CA 92152

Dr. A.L. Slafkosky 1
Scientific Advisor
Commandant of the Marine Corps (Code RD-1)
Washington, D.C. 20380

Mr. E.H. Gleissner 1
Naval Ship Research & Development Center.
Computation & Mathematics Dept.
Bethesda, MD 20084

Capt. Grace M. Hopper 1
NAICOM/MIS Planning Branch (OP-916D)
Office of Chief of Naval Operations
Washington, D.C. 20350

Mr. Kin B. Thompson 1
NAVDAC 33
Washington Navy Yard
Washington, D.C. 20374

Advanced Research Projects Agency 1
Information Processing Techniques
1400 Wilson Boulevard
Arlington, VA 22209

Capt. Richard L. Martin, USN 1
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York 09501